EXHIBIT 11

EXHIBIT D

<u>Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)</u>

Chart Detailing Defendant's Infringement of U.S. Patent No. 10,353,811

Wapp Tech Ltd. & Wapp Tech Corp. v. J.P. Morgan Chase Bank, N.A..
Case No. 4:23-cv-1137-ALM (E.D. Tex.)

The Accused Instrumentalities include tools from Google used to develop applications for Android mobile devices, including Android Studio, Android Emulator, Android Virtual Devices, and Android Profiler tools.

Based on the information presently available to them, Plaintiffs Wapp Tech Limited Partnership and Wapp Tech Corp. ("Wapp" or "Plaintiffs") are informed and believe that Defendant directly and indirectly infringes U.S. Patent No. 10,353,811 (the "'811 Patent"). Defendant directly infringes the '811 Patent when its employees, agents, and/or representatives use the Accused Instrumentalities to develop applications for mobile devices. Upon information and belief, to the extent Defendant uses third parties in the development process, Defendant indirectly infringes the '811 Patent by actively inducing the direct infringement of third parties contracted to use the Accused Instrumentalities to develop applications for mobile devices on Defendant's behalf.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

Table of Contents

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[A] A non-transitory, computer-readable medium comprising software instructions for developing an application to be run on a mobile device, wherein the software instructions, when executed, cause a computer to:

Claim 1

1[A] A non-transitory, computer-readable medium comprising software instructions for developing an application to be run on a mobile device, wherein the software instructions, when executed, cause a computer to:

The hard drive of a laptop or desktop running Android Studio is a non-transitory, computer-readable medium comprising software instructions for developing an application to be run on a mobile device, such as an Android-based phone, tablet, or watch.

Android Studio is software comprising software instructions for developing an application to be run on a mobile device. Defendant develops mobile banking applications through its use of the Accused Instrumentalities by writing source code for the application, compiling that source code, and testing the code. Android Studio is an Integrated Development Environment (IDE) for developing applications based on the Android operating system. Developers use Android Studio to develop applications by writing source code and compiling that source code into programs that will run on Android devices. The Android operating system runs on various mobile devices, including smartphones, tablets, and wearables. Android Studio includes "[a] fast and feature-rich emulator" and "[a] unified environment where you can develop for all Android devices."

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[A] A non-transitory, computer-readable medium comprising software instructions for developing an application to be run on a mobile device, wherein the software instructions, when executed, cause a computer to:

Meet Android Studio

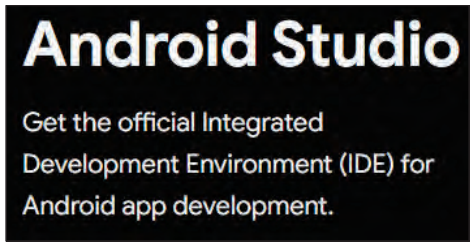
Android Studio is the official Integrated Development Environment (IDE) for Android app development. Based on the powerful code editor and developer tools from IntelliJ IDEA, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Live Edit to update composables in emulators and physical devices in real time
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

https://developer.android.com/studio/intro (last visited 4/6/2024).

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[A] A non-transitory, computer-readable medium comprising software instructions for developing an application to be run on a mobile device, wherein the software instructions, when executed, cause a computer to:


Android Studio

Get the official Integrated Development Environment (IDE) for Android app development.

https://developer.android.com/studio (last visited 4/6/2024).

Android Studio includes a code editor for developing mobile device applications.


Intelligent code editor

Write better code, work faster, and be more productive with an intelligent code editor that provides code completion for Kotlin, Java, and C/C++ programing languages. Moreover, when editing Jetpack Compose you can see your code changes reflected immediately with Live Edit.

https://developer.android.com/studio (last visited 4/6/2024).

Android Studio includes an emulator for developing mobile device applications. The emulator allows application authors to "test [their] application on a variety of Android devices."

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[A] A non-transitory, computer-readable medium comprising software instructions for developing an application to be run on a mobile device, wherein the software instructions, when executed, cause a computer to:

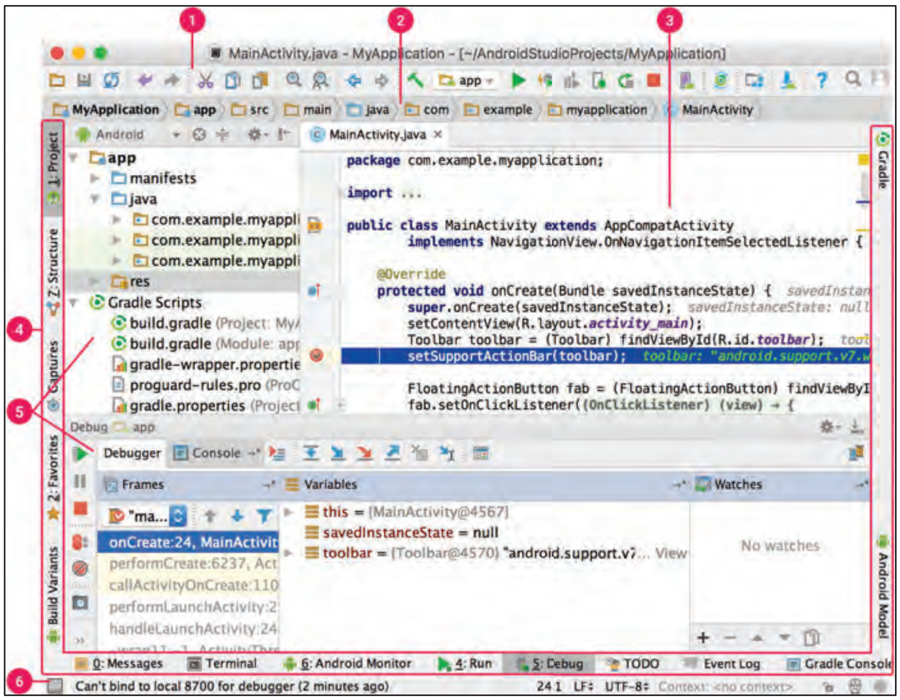
> **Easily emulate any device**
>
> The Android Emulator lets you to test your application on a variety of Android devices. Unlock the full potential of your apps by using responsive layouts that adapt to fit phones, tablets, foldables, Wear OS, TV and ChromeOS devices.

https://developer.android.com/studio (last visited 4/6/2024).

The following figure illustrates the Android Studio main window that is used for viewing and editing source code files.

<u>Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)</u>

1[A] A non-transitory, computer-readable medium comprising software instructions for developing an application to be run on a mobile device, wherein the software instructions, when executed, cause a computer to:



https://developer.android.com/studio/intro (last visited 4/6/2024). The editor window (#3) is used to create and modify source code, which is part of developing an application. The remaining windows are described below:

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[A] A non-transitory, computer-readable medium comprising software instructions for developing an application to be run on a mobile device, wherein the software instructions, when executed, cause a computer to:

1. **Toolbar:** Carry out a wide range of actions, including running your app and launching Android tools.
2. **Navigation bar:** Navigate through your project and open files for editing. It provides a more compact view of the structure visible in the **Project** window.
3. **Editor window:** Create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.
4. **Tool window bar:** Use the buttons on the outside of the IDE window to expand or collapse individual tool windows.
5. **Tool windows:** Access specific tasks like project management, search, version control, and more. You can expand them and collapse them.
6. **Status bar:** Display the status of your project and the IDE itself, as well as any warnings or messages.

https://developer.android.com/studio/intro/user-interface (last visited 4/6/2024). Android Studio's software authoring interface includes a number of different "windows" or tools that are available to the application author throughout the authoring process.

On information and belief, Defendant uses Android Studio to develop mobile banking applications for its business, for example—Chase Mobile. While Chase Mobile is identified as an example application, the contentions detailed in this chart apply to all mobile application development done by or on behalf of Defendant using Android Studio. On information and belief, Defendant's development of mobile banking applications includes using the features detailed throughout this document.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[B] display a list of a plurality of mobile device models from which a user can select,

1[B] display a list of a plurality of mobile device models from which a user can select,

Android Studio displays a list of mobile device models (such as phones and tablets) from which a user can select to emulate/simulate how the application will run on that model. The application being developed can then be deployed on a model that is specific to that device.

Android Studio's Device Manager[1] allows a user to create a new virtual device, or an Android Virtual Device (AVD). An AVD specifies the "hardware characteristics" of a device to be used for emulation/simulation. Each AVD specifies the resources of the emulated/simulated device. These AVDs are created and managed in Android Studio using the Android Device manager.

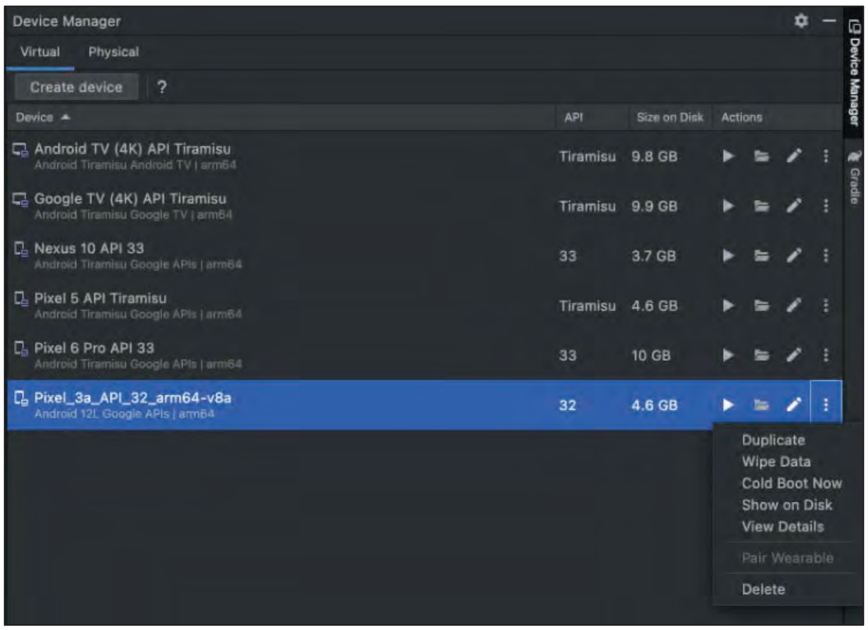
> ### Create an Android Virtual Device
>
> Each instance of the Android Emulator uses an *Android virtual device (AVD)* to specify the Android version and hardware characteristics of the simulated device. To effectively test your app, create an AVD that models each device your app is designed to run on. To create an AVD, see Create and manage virtual devices.
>
> Each AVD functions as an independent device with its own private storage for user data, SD card, and so on. By default, the emulator stores the user data, SD card data, and cache in a directory specific to that AVD. When you launch the emulator, it loads the user data and SD card data from the AVD directory.

https://developer.android.com/studio/run/emulator (last visited 4/6/2024).

[1] Prior to the Bumblebee release, Device Manager was referred to as the Android Virtual Device Manager.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[B] display a list of a plurality of mobile device models from which a user can select,



https://developer.android.com/studio/run/managing-avds (last visited 4/6/2024) (illustrating Android Device Manager).

Once an Android Virtual Device is created based on a particular hardware profile, its operating properties can be specified. All of these properties are simulated/emulated during operation within the Android Emulator.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[B] display a list of a plurality of mobile device models from which a user can select,

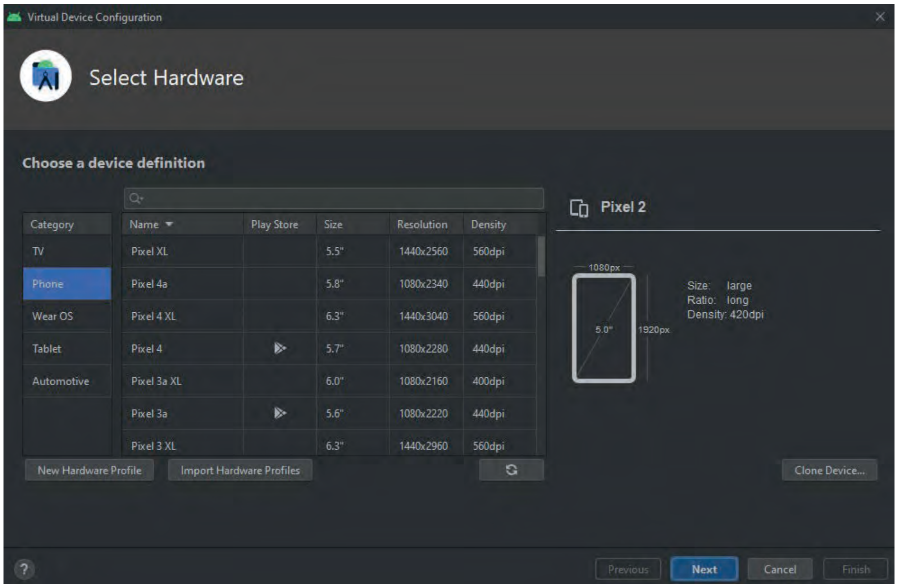
> # Create and manage virtual devices
>
> An Android Virtual Device (AVD) is a configuration that defines the characteristics of an Android phone, tablet, Wear OS, Android TV, or Automotive OS device that you want to simulate in the Android Emulator. The Device Manager is a tool you can launch from Android Studio that helps you create and manage AVDs.

https://developer.android.com/studio/run/managing-avds (last visited 4/6/2024).

Before emulation/simulation, the user must create an Android Virtual Device (AVD) based on a hardware profile. Android Studio provides a set of hardware profiles to select from and also permits the user to create custom hardware profiles. As illustrated below, the hardware profiles are divided into different categories (e.g., phone, table); each category includes one or more hardware profiles for that category (e.g., Pixel XL, Pixel 4A) that defines base characteristics of the device model (e.g., screen size, memory, cameras, sensors). The hardware profiles are a list of a plurality of mobile device models from which a user select. Each of these device models can be used to create an Android Virtual Device to run in the Android Emulator.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[B] display a list of a plurality of mobile device models from which a user can select,



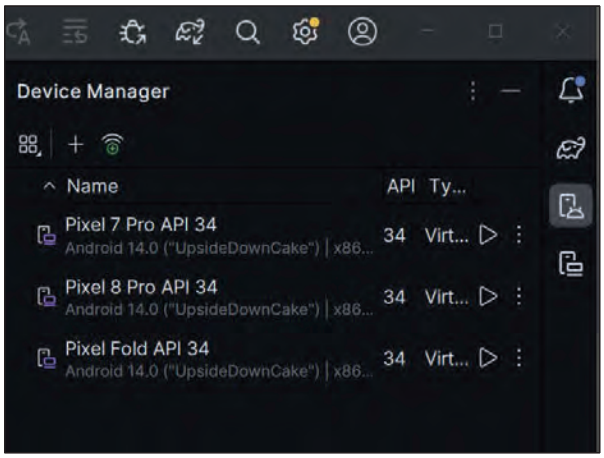
Screenshot from Android Studio Arctic Fox[2]: Virtual Device Configuration.

Android Device Manager also displays a list of mobile device models from which a user can select—namely, AVDs that are created based on the hardware profiles detailed above. An example list of AVDs in Device Manager is illustrated below:

[2] New versions of Android Studio are released on a regular basis. The user interface may differ slightly between different versions; however, the functionality identified in this chart exists in Android Studio versions from 2017 to the present. To the extent the functionality in different versions of Android Studio changes in a meaningful way regarding the infringement read, those changes will be noted in the chart.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[B] display a list of a plurality of mobile device models from which a user can select,



Screenshot from Android Studio Iguana: Device Manager.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[C] wherein each model includes one or more characteristics indicative of a corresponding mobile device;

1[C] wherein each model includes one or more characteristics indicative of a corresponding mobile device;

Each model (hardware profile or AVD) includes one or more characteristics indicative of a corresponding mobile device, such as screen resolution and button availability. Accordingly, one or more characteristics of these selected mobile device types are simulated.

For example, a Pixel 4 will have a smaller screen than a Pixel 4 XL.

Screenshot from Android Studio Arctic Fox: Virtual Device Configuration: Select Hardware (showing properties of Pixel 4 XL).

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

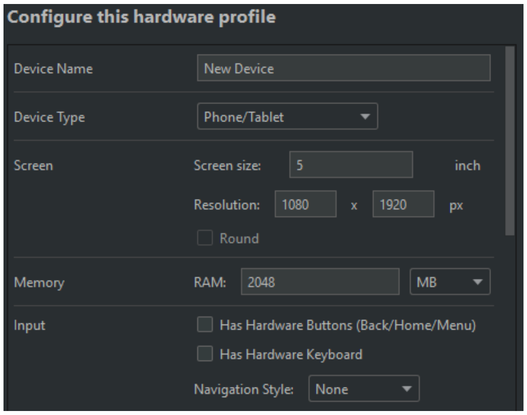
1[C] wherein each model includes one or more characteristics indicative of a corresponding mobile device;

Screenshot from Android Studio Arctic Fox: Virtual Device Configuration: Select Hardware (showing properties of Pixel 4).

Each hardware profile includes multiple characteristics, as illustrated below:

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

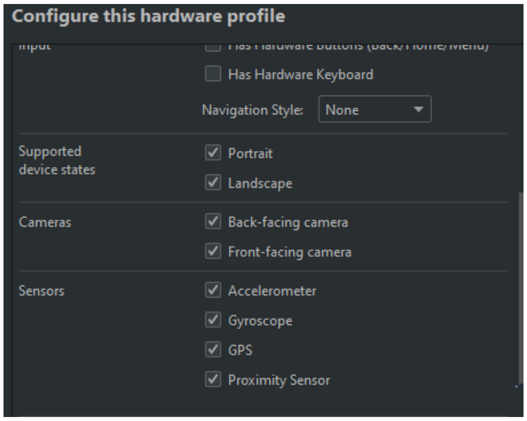
1[C] wherein each model includes one or more characteristics indicative of a corresponding mobile device;



Screenshot from Android Studio Arctic Fox: Virtual Device Configuration: Create Hardware Profile (showing hardware profile characteristics).

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[C] wherein each model includes one or more characteristics indicative of a corresponding mobile device;



Android Studio: Virtual Device Configuration: Create Hardware Profile (showing hardware profile characteristics). Each Android Virtual Device is based on a selected hardware profile and inherits these same hardware profile characteristics.

Each Android Virtual Device has a profile that includes a number of properties defining the characteristics of the AVD to emulate/simulate. The "hardware profile properties" include:

- Device Name
- Device Type
- Screen: Screen Size
- Screen: Screen Resolution
- Screen: Round
- Memory: RAM
- Input: Has Hardware Buttons (Back/Home/Menu)
- Input: Has Hardware Keyboard
- Input: Navigation Style
- Supported Device States

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[C] wherein each model includes one or more characteristics indicative of a corresponding mobile device;

- Cameras
- Sensors: Accelerometer
- Sensors: Gyroscope
- Sensors: GPS
- Sensors: Proximity Sensor
- Default Skin.

See https://developer.android.com/studio/run/managing-avds (last visited 4/6/2024) (listing and describing hardware profile properties). Additionally, an AVD has AVD properties that also control the manner in which the AVD performs during emulation/simulation. The AVD Properties include:

- AVD Name
- AVD ID (Advanced)
- Hardware Profile
- System Image
- Startup Orientation
- Camera (Advanced)
- Network: Speed (Advanced)
- Network: Latency (Advanced)
- Emulated Performance: Graphics
- Emulated Performance: Boot option (Advanced)
- Emulated Performance: Multi-Core CPU (Advanced)
- Memory and Storage: RAM (Advanced)
- Memory and Storage: VM Heap (Advanced)
- Memory and Storage: Internal Storage (Advanced)
- Memory and Storage: SD Card (Advanced)
- Device Frame: Enable Device Frame
- Custom Skin Definition (Advanced)
- Keyboard: Enable Keyboard Input (Advanced)

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[C] wherein each model includes one or more characteristics indicative of a corresponding mobile device;

See https://developer.android.com/studio/run/managing-avds (last visited 4/6/2024) (listing and describing AVD properties). Each of the above-mentioned properties represent characteristics indicative of the mobile device to which the model corresponds.

Some hardware profile properties and AVD properties can also be controlled via the command-line options available for the Android Emulator. Examples of the command-line options permitting control of hardware characteristics of the AVD device during simulation are provided below.

`-memory size`	Specify the physical RAM size from 128 to 4096 MBs. For example: `$ emulator @Nexus_5X_API_23 -memory 2048`

https://developer.android.com/studio/run/emulator-commandline (last visited 4/6/2024) (showing options for controlling the RAM size of the emulated/simulated device).

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[C] wherein each model includes one or more characteristics indicative of a corresponding mobile device;

`-screen mode`	Set emulated touch screen mode. For example: `$ emulator @Nexus_5X_API_23 -screen no-touch` *mode* can be any of the following values: • `touch` - Emulate a touch screen (default). • `multi-touch` - Emulate a multi-touch screen. • `no-touch` - Disable touch and multi-touch screen emulation.

https://developer.android.com/studio/run/emulator-commandline (last visited 4/6/2024) (showing options for controlling the touch capabilities to emulate/simulate for the device screen).

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[D] simulate at least one of the one or more characteristics indicative of the mobile device corresponding to the selected mobile device model;

1[D] simulate at least one of the one or more characteristics indicative of the mobile device corresponding to the selected mobile device model;

Android Studio's Emulator simulates characteristics indicative of the mobile device corresponding to the selected mobile device model, such as screen size, screen resolution, RAM, CPU, and the presence or absence of a button. *See* 1[C] (listing the hardware profile characteristics for an AVD).

Each of the above-mentioned properties (*see* 1[C]) are emulated/simulated by Android Emulator when running the device profile specified for that particular AVD. The hardware profile properties and the AVD properties represent characteristics indicative of the mobile device corresponding to the selected mobile device model. When the Emulator runs a particular AVD, these characteristics are emulated/simulated and are indicative of the performance of the emulated/simulated device when running the application.

1[E] simulate one or more characteristics indicative of a network on which the mobile device corresponding to the selected mobile device model can operate;

Android Studio simulates characteristics indicative of a network on which the mobile device corresponding to the selected mobile device model can operate.

The Android Emulator is programmed to emulate/simulate Android devices on a computer without actually possessing the physical device. The Android Emulator emulates/simulates "different network speeds . . . ."

> The Android Emulator simulates Android devices on your computer so that you can test your application on a variety of devices and Android API levels without needing to have each physical device. The emulator offers these advantages:
>
> - **Flexibility**: In addition to being able to simulate a variety of devices and Android API levels, the emulator comes with predefined configurations for various Android phone, tablet, Wear OS, and Android TV devices.
> - **High fidelity**: The emulator provides almost all the capabilities of a real Android device. You can simulate incoming phone calls and text messages, specify the location of the device, simulate different network speeds, simulate rotation and other hardware sensors, access the Google Play Store, and much more.
> - **Speed**: Testing your app on the emulator is in some ways faster and easier than doing so on a physical device. For example, you can transfer data faster to the emulator than to a device connected over USB.
>
> In most cases, the emulator is the best option for your testing needs. This page covers the core emulator functionalities and how to get started with it.

https://developer.android.com/studio/run/emulator (last visited 4/6/2024).

The Android Emulator is programmed to simulate/emulate a plurality of network characteristics, such as network speed (upload/download) and network latency.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[E] simulate one or more characteristics indicative of a network on which the mobile device corresponding to the selected mobile device model can operate;

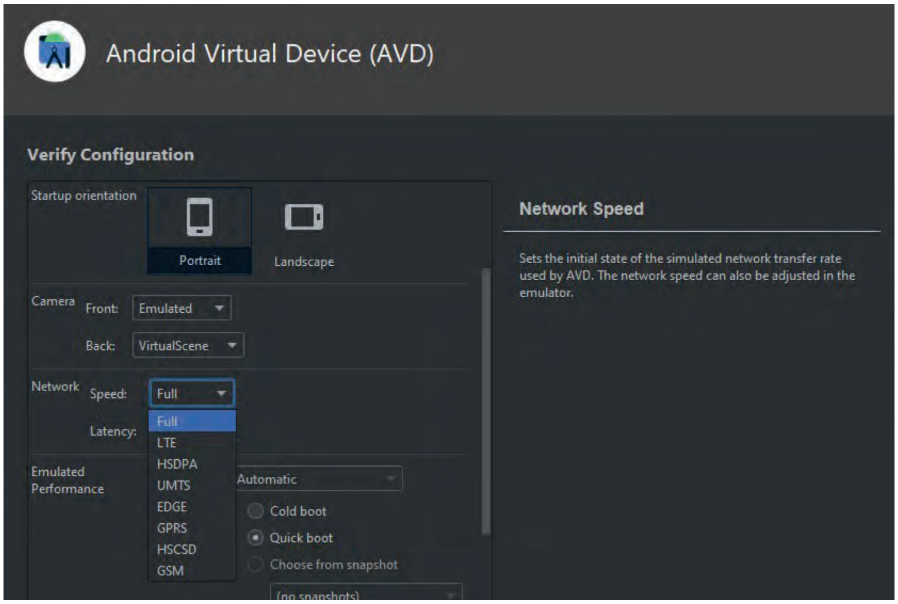
> The emulator supports network throttling as well as higher connection latencies. You can define it either through the skin configuration or with the `-netspeed` and `-netdelay` options.

https://developer.android.com/studio/run/emulator-commandline (last visited 4/30/2024).

The following figures show screenshots of the configuration screens for an Android Virtual Device. The first figure shows how an AVD can have a specific network speed associated with it, the speed options including: Full, LTE, HSDPA, UMTS, EDGE, GPRS, HSCSD, GSM.

<u>Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)</u>

1[E] simulate one or more characteristics indicative of a network on which the mobile device corresponding to the selected mobile device model can operate;

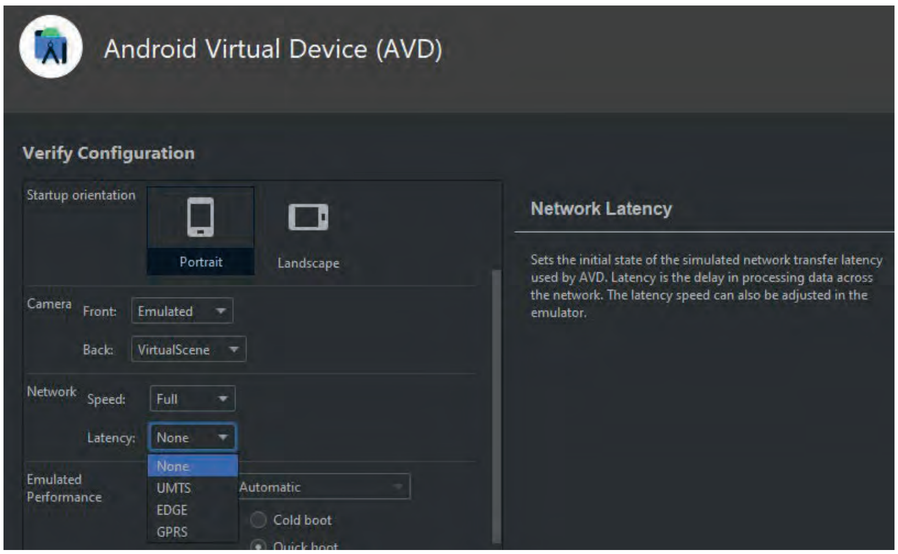


Screenshot from Android Studio Arctic Fox: Android Virtual Device (showing Network Speed options: Full, LTE, HSDPA, UMTS, EDGE, GPRS, HSCSD, GSM).

The following screenshot shows how an AVD can have a specific network latency associated with it, the latency options including: None, UMTS, EDGE, GPRS.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[E] simulate one or more characteristics indicative of a network on which the mobile device corresponding to the selected mobile device model can operate;



Screenshot from Android Studio Arctic Fox: Android Virtual Device (showing Network Latency options).

The network characteristics can also be controlled via the command-line options available for the Android Emulator, as illustrated below.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[E] simulate one or more characteristics indicative of a network on which the mobile device corresponding to the selected mobile device model can operate;

`-netdelay` *`delay`*	Set network latency emulation to one of the following *`delay`* values in milliseconds: • `gsm` - GSM/CSD (min 150, max 550). • `hscsd` - HSCSD (min 80, max 400). • `gprs` - GPRS (min 35, max 200). • `edge` - EDGE/EGPRS (min 80, max 400). • `umts` - UMTS/3G (min 35, max 200). • `hsdpa` - HSDPA (min 0, max 0). • `lte` - LTE (min 0, max 0). • `evdo` - EVDO (min 0, max 0). • `none` - No latency, the default (min 0, max 0). • *`num`* - Specify exact latency. • *`min`*:*`max`* - Specify individual minimum and maximum latencies. For example: `$ emulator @Nexus_5X_API_23 -netdelay gsm`

https://developer.android.com/studio/run/emulator-commandline (last visited 4/7/2024) (showing options for setting network latency properties).

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[E] simulate one or more characteristics indicative of a network on which the mobile device corresponding to the selected mobile device model can operate;

`-netspeed` *speed*	Set the network speed emulation. Specify the maximum network upload and download speeds with one of the following *speed* values in kbps: • `gsm` - GSM/CSD (up: 14.4, down: 14.4). • `hscsd` - HSCSD (up: 14.4, down: 57.6). • `gprs` - GPRS (up: 28.8, down: 57.6). • `edge` - EDGE/EGPRS (up: 473.6, down: 473.6). • `umts` - UMTS/3G (up: 384.0, down: 384.0). • `hsdpa` - HSDPA (up: 5760.0, down: 13,980.0). • `lte` - LTE (up: 58,000, down: 173,000). • `evdo` - EVDO (up: 75,000, down: 280,000). • `full` - No limit, the default (up: 0.0, down: 0.0). • *num* - Specify both upload and download speed. • *up*:*down* - Specify individual up and down speeds.

https://developer.android.com/studio/run/emulator-commandline (last visited 4/7/2024) (showing options for setting network speed properties).

These network characteristics (e.g., speed and latency) are indicative of a network on which the mobile device corresponding to the selected mobile device model can operate.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[F] monitor utilization of a plurality of resources over time as the application is running;

1[F] monitor utilization of a plurality of resources over time as the application is running;

Android Studio monitors utilization of a plurality of resources over time as the application is running. The Android Emulator, Profilers, App Inspectors, and System trace monitor the utilization of various resources over time as the mobile application is running on the selected AVD being emulated/simulated.

Android Studio includes a number of profiling tools that support application development, allowing a software author to monitor the resources of the Android device or AVD that are used by the application while executing on the device. These profiling tools have corresponding display windows.

The Android profiling tools for Arctic Fox and relevant prior releases include: (1) CPU Profiler; (2) Memory Profiler; (3) Network Profilers; and (4) Energy Profiler. Starting with the Bumblebee release, the Network Profiler was moved to App Inspection and renamed the Network Inspector, as detailed further below.

These profiling tools are used for "[f]ixing performance problems [which] involves identifying areas in which your app makes inefficient use of resources such as the CPU, memory, graphics, network, and the device battery . . . . Android studio offers several profiling tools to help find and visualize potential problems."

Profile your app performance

An app is considered to have poor performance if it responds slowly, shows choppy animations, freezes, or consumes too much power. Fixing performance problems involves identifying areas in which your app makes inefficient use of resources such as the CPU, memory, graphics, network, and the device battery. To find and fix these problems, use the profiling and benchmarking tools and techniques described in this topic.

Android Studio offers several profiling tools to help find and visualize potential problems:

- **CPU profiler**: This tool helps track down runtime performance issues.
- **Memory profiler**: This tool helps track memory allocations.
- **Network profiler**: This tool monitors network traffic usage.
- **Energy profiler**: This tool tracks energy usage, which can contribute to battery drain

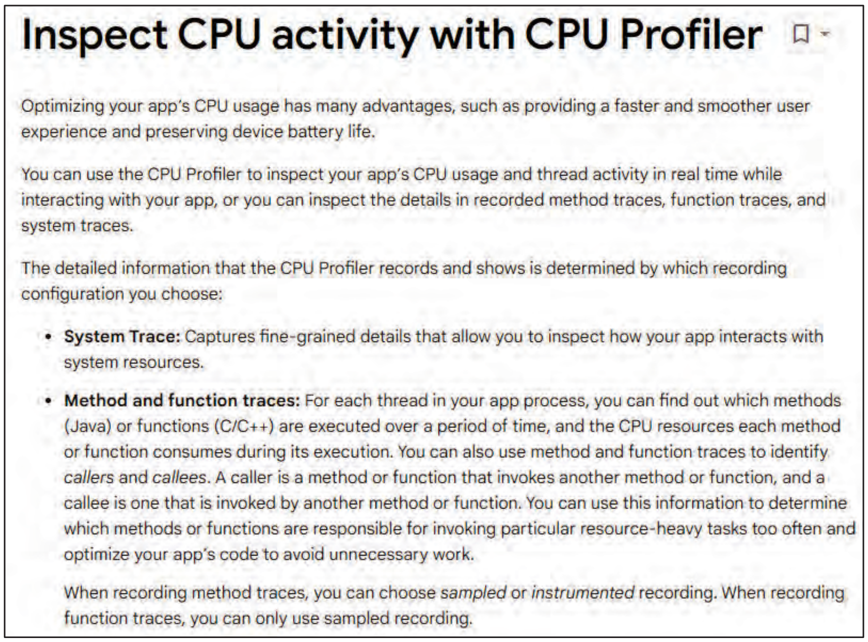
https://developer.android.com/studio/profile (last visited 1/6/2022).

<u>Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)</u>

1[F] monitor utilization of a plurality of resources over time as the application is running;

Android Studio allows the application author to profile the CPU, memory usage, network activity, and energy use of the mobile device while executing an application. Each profiler is detailed below.

The **<u>CPU Profiler</u>** is used to monitor CPU usage and availability, and it helps track down runtime performance issues. It can be used to "inspect your app's CPU usage and thread activity in real time while interacting with your app . . . ."



Inspect CPU activity with CPU Profiler

Optimizing your app's CPU usage has many advantages, such as providing a faster and smoother user experience and preserving device battery life.

You can use the CPU Profiler to inspect your app's CPU usage and thread activity in real time while interacting with your app, or you can inspect the details in recorded method traces, function traces, and system traces.

The detailed information that the CPU Profiler records and shows is determined by which recording configuration you choose:

- **System Trace:** Captures fine-grained details that allow you to inspect how your app interacts with system resources.
- **Method and function traces:** For each thread in your app process, you can find out which methods (Java) or functions (C/C++) are executed over a period of time, and the CPU resources each method or function consumes during its execution. You can also use method and function traces to identify *callers* and *callees*. A caller is a method or function that invokes another method or function, and a callee is one that is invoked by another method or function. You can use this information to determine which methods or functions are responsible for invoking particular resource-heavy tasks too often and optimize your app's code to avoid unnecessary work.

  When recording method traces, you can choose *sampled* or *instrumented* recording. When recording function traces, you can only use sampled recording.

https://developer.android.com/studio/profile/cpu-profiler (last visited 4/7/2024) (detailing the CPU Profiler). The CPU Profiler displays the executed application's CPU usage and thread activity via timelines, as illustrated below.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[F] monitor utilization of a plurality of resources over time as the application is running;

Figure 1. Timelines in the CPU Profiler.

https://developer.android.com/studio/profile/cpu-profiler (last visited 4/7/2024) (illustrating the CPU Profiler timelines).

Referring to the numbers in the figure above, Number 1 illustrates the Event Timeline, which "[s]hows the activities in your app as they transition through different states in their lifecycle, and indicates user interactions with the device, including screen rotation events." *Id.* Number 2 illustrates the CPU Timeline, which "[s]hows real-time CPU usage of your app—as a percentage of total available CPU time—and the total number of threads your app is using. The timeline also shows the CPU usage of other processes (such as system processes or other apps), so you can compare it to your app's usage. You can inspect historical CPU usage data by moving your mouse along the horizontal axis of the timeline." *Id.* Finally, Number 3 indicates the Thread Activity Timeline, which "[l]ists each thread that belongs to your app process and indicates its activity along a timeline using the colors listed below." *Id.*

<u>Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)</u>

1[F] monitor utilization of a plurality of resources over time as the application is running;

The CPU Timeline (Number 2) shows the CPU usage of the application during execution. In particular, the lighter green portions of the CPU timeline illustrate the percentage of CPU resources used by the application at any given point in time. The darker green/gray portion of the timeline shows the CPU resources consumed by other components on the device (e.g., system processes or other applications). Finally, the remaining dark/black portion of the timeline shows the amount of CPU resources not currently used.

The **<u>Memory Profiler</u>** is used to monitor memory usage by the application. It assists with detecting unwanted or unnecessary memory consumption, including memory leaks and memory churn.

> **Inspect your app's memory usage with Memory Profiler**
>
> The Memory Profiler is a component in the Android Profiler that helps you identify memory leaks and memory churn that can lead to stutter, freezes, and even app crashes. It shows a realtime graph of your app's memory use and lets you capture a heap dump, force garbage collections, and track memory allocations.

https://developer.android.com/studio/profile/memory-profiler (last visited 2/7/2024) (describing the Memory Profiler). An example view of the Memory Profiler is provided below:

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[F] monitor utilization of a plurality of resources over time as the application is running;

Figure 1. The Memory Profiler

https://developer.android.com/studio/profile/memory-profiler (last visited 4/7/2024) (illustrating the Memory Profiler). The memory legend near the top illustrates the amount of memory consumed or utilized by the application (e.g., Total, Java, Native, Graphics, Stack, Code, Others). In addition, the Allocated component (represented in the graph by a white dotted line) indicates the amount of allocated memory for the application. In this way, the Memory Profiler provides information about the unallocated and allocated memory resources utilized by an application for a given AVD configuration.

The **Network Profiler** allows the application author to monitor network connections and data exchanges by the mobile application.

Inspect network traffic with Network Profiler

The Network Profiler displays realtime network activity on a timeline, showing data sent and received, as well as the current number of connections. This lets you examine how and when your app transfers data, and optimize the underlying code appropriately.

https://developer.android.com/studio/profile/network-profiler (last visited 1/6/2022).

As illustrated below, the Network Profiler shows the receiving network speed, the sending network speed, and latency.

<u>Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)</u>

1[F] monitor utilization of a plurality of resources over time as the application is running;

https://developer.android.com/studio/profile/network-profiler (last visited 7/27/2021).

The Connection View provides additional information about the network transmissions, including transmission duration and timing, which is related to network latency.

> - **Connection View**: Lists files that were sent or received during the selected portion of the timeline across all of your app's CPU threads. For each request, you can inspect the size, type, status, and transmission duration. You can sort this list by clicking any of the column headers. You also see a detailed breakdown of the selected portion of the timeline, showing when each file was sent or received.

https://developer.android.com/studio/profile/network-profiler (last visited 7/27/2021).

The Thread View displays the network activity for each of the application's CPU threads, illustrating the receiving network speed, the sending network speed, and latency.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[F] monitor utilization of a plurality of resources over time as the application is running;

Figure 2. Inspecting network requests by app thread in the **Thread View**

https://developer.android.com/studio/profile/network-profiler (last visited 1/7/2022). The blue line in the network timeline shows the receiving rate of data flow, and the orange line shows the sending rate of data flow for the application. Each of these translate to bandwidth resources used by the application. The dotted line indicates the number of connections and the maximum data transfer rate employed by the application.

Additionally, as detailed above, the AVD is configured with network throttling properties that limit the bandwidth—both upload and download speeds—that can be used by the application.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[F] monitor utilization of a plurality of resources over time as the application is running;

`-netspeed` *speed*	Set the network speed emulation. Specify the maximum network upload and download speeds with one of the following *speed* values in kbps: • `gsm` - GSM/CSD (up: 14.4, down: 14.4). • `hscsd` - HSCSD (up: 14.4, down: 57.6). • `gprs` - GPRS (up: 28.8, down: 57.6). • `edge` - EDGE/EGPRS (up: 473.6, down: 473.6). • `umts` - UMTS/3G (up: 384.0, down: 384.0). • `hsdpa` - HSDPA (up: 5760.0, down: 13,980.0). • `lte` - LTE (up: 58,000, down: 173,000). • `evdo` - EVDO (up: 75,000, down: 280,000). • `full` - No limit, the default (up: 0.0, down: 0.0). • *num* - Specify both upload and download speed. • *up*:*down* - Specify individual up and down speeds.

https://developer.android.com/studio/run/emulator-commandline (last visited 4/6/2024) (showing the upload and download maximums for different "network speed" settings provided in the AVD configuration).

The **Energy Profiler** allows the author to monitor energy consumption by the application.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[F] monitor utilization of a plurality of resources over time as the application is running;

Inspect energy use with Energy Profiler

The Energy Profiler helps you to find where your app uses more energy than necessary.

The Energy Profiler monitors the use of the CPU, network radio, and GPS sensor, and it displays a visualization of how much energy each of these components uses. The Energy Profiler also shows you occurrences of system events (wake locks, alarms, jobs, and location requests) that can affect energy consumption.

The Energy Profiler does not directly measure energy consumption. Rather, it uses a model that estimates the energy consumption for each resource on the device.

https://developer.android.com/studio/profile/energy-profiler (last visited 4/7/2024). An example view of the Energy Profiler is provided below:

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[F] monitor utilization of a plurality of resources over time as the application is running;

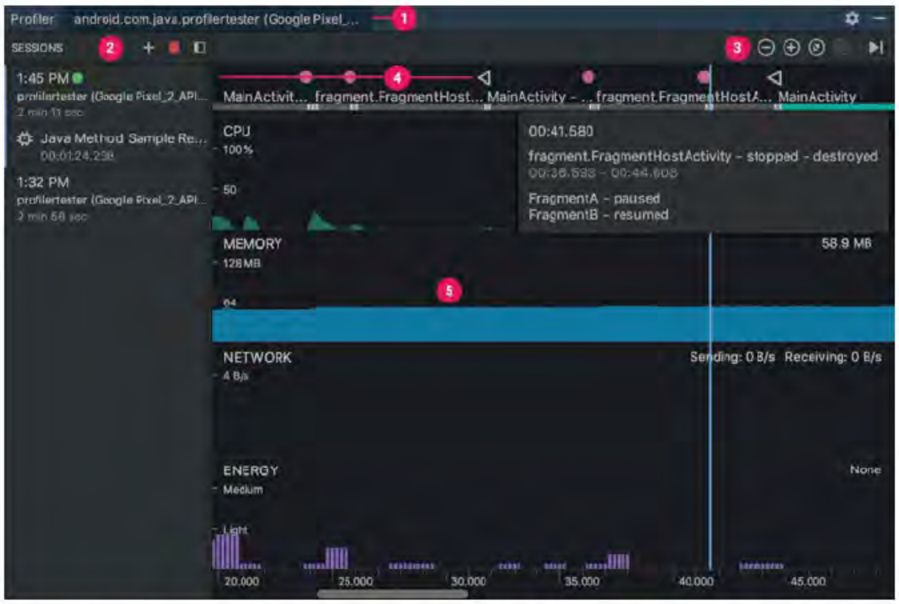
Figure 1. Timelines in the Energy Profiler.

https://developer.android.com/studio/profile/energy-profiler (last visited 4/7/2024). Three timelines are illustrated in this Energy Profiler view. The first (1) is the Event Timeline, which "[s]hows the activities in your app as they transition through different states in their lifecycle. This timeline also indicates user interactions with the device, including screen rotation events." *Id.* The second (2) is the Energy Timeline, which "[s]hows estimated energy consumption of your app." *Id.* And the third (3) is the System Time, which "[i]ndicates system events that may affect energy consumption." *Id.* By moving your mouse over the timelines, you can "see a breakdown of energy use by CPU, network, and location (GPS) resources . . . ." *Id.*

In addition to the exemplary profile display windows illustrated above, Android Studio Arctic Fox (and earlier versions) supports display of profile windows for all four profilers discussed above:

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[F] monitor utilization of a plurality of resources over time as the application is running;

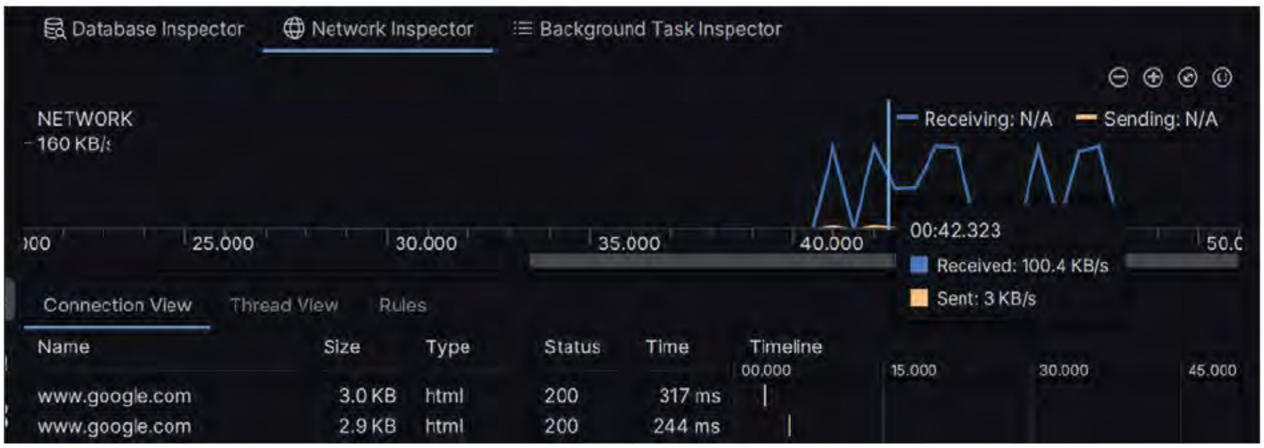


https://developer.android.com/studio/profile/android-profiler (last visited 7/27/2021) (illustrating the CPU, Memory, Network, and Energy profile displays).

Starting with the Android Studio Bumblebee release, the Network Profiler was moved to the App Inspection component of Android Studio and renamed Network Inspector. *See* https://developer.android.com/studio/releases/past-releases/as-bumblebee-release-notes (last visited 4/20/2024). The Network Inspector view is shown below:

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[F] monitor utilization of a plurality of resources over time as the application is running;



Screenshot from Android Studio Iguana (showing Network Inspector). The Network Inspector view shows the receiving network speed and the sending network speed. And it includes the Connection View and Thread View discussed above for the Network Profiler.

The CPU and Memory Profilers are available in post-Arctic Fox versions of Android Studio, as detailed above for Arctic Fox.

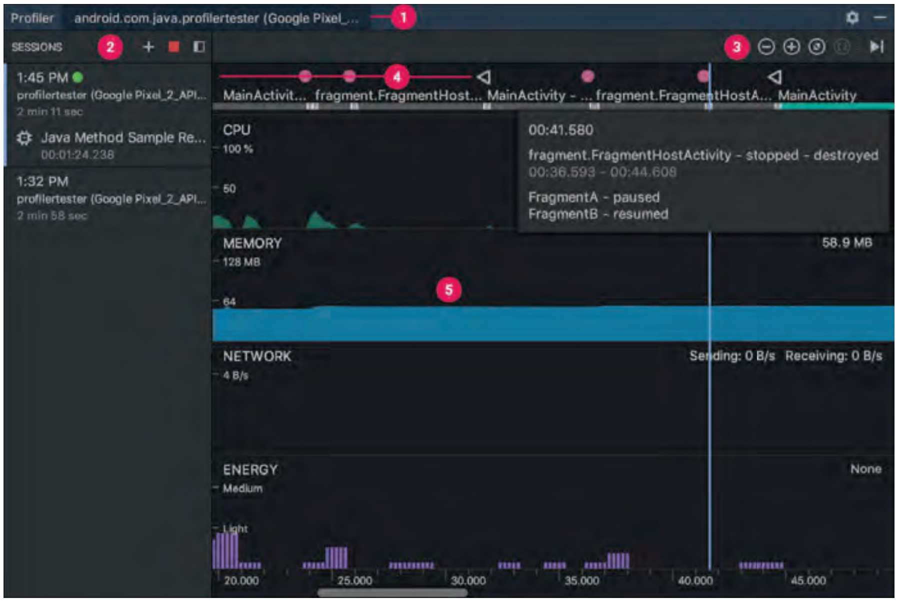
Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[G] display simultaneously two or more graphical images of the application's resource utilization, wherein each graphical image relates to a different resource;

1[G] display simultaneously two or more graphical images of the application's resource utilization, wherein each graphical image relates to a different resource;

Android Studio displays simultaneously two or more graphical images of the application's resource utilization, wherein each graphical image relates to a different resource.

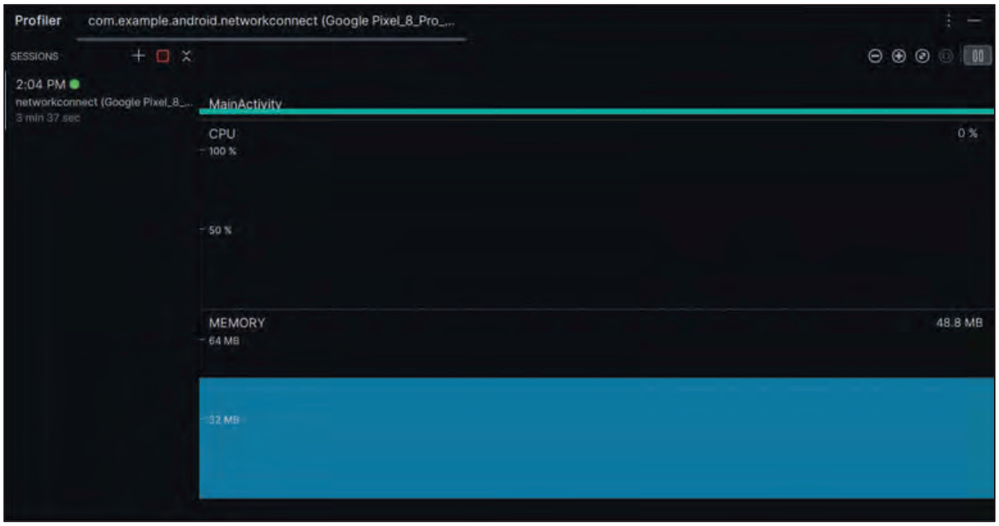
The profiler view discussed above (*see* 1[F]) can display all of the resource utilization graphs simultaneously. For example, in the screenshot below CPU, Memory, Network, and Energy resource utilization are shown simultaneously.



https://developer.android.com/studio/profile/android-profiler (last visited 7/27/2021). Newer versions of the Android Profiler (post Arctic Fox) continue to show multiple resource utilization graphs.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[G] display simultaneously two or more graphical images of the application's resource utilization, wherein each graphical image relates to a different resource;



Screenshot from Android Studio Iguana: Profiler (showing CPU and Memory Profiler utilization displayss).

Additionally, the Network Profiler (and Network Inspector in newer versions of Android Studio) shows both the receiving network speed and the sending network speed, illustrating two different resources (upload vs. download bandwidth) simultaneously with different graphical representations.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[G] display simultaneously two or more graphical images of the application's resource utilization, wherein each graphical image relates to a different resource;

https://developer.android.com/studio/profile/network-profiler (last visited 7/27/2021).

The Thread View displays the network activity for each of the application's CPU threads, illustrating the receiving network speed, the sending network speed, and latency.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[G] display simultaneously two or more graphical images of the application's resource utilization, wherein each graphical image relates to a different resource;

Figure 2. Inspecting network requests by app thread in the **Thread View**

https://developer.android.com/studio/profile/network-profiler (last visited 1/7/2022). The blue line in the network timeline is a graphical image that shows the receiving rate of data flow, and the orange line is a graphical image that shows the sending rate of data flow for the application. Each of these translate to bandwidth resources used by the application. Further, the Thread View is another graphical image that also shows the times during which sending and receiving for each thread of the application occurs.

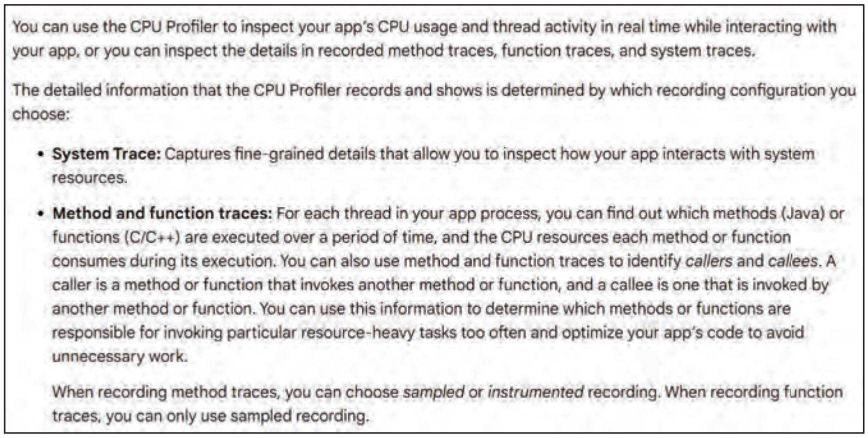
Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[H] correspond the utilization of a specific displayed resource at a given time with one or more functions of the application responsible for that utilization.

1[H] correspond the utilization of a specific displayed resource at a given time with one or more functions of the application responsible for that utilization.

Android Studio corresponds the utilization of a specific displayed resources at a given time with one or more functions of the application responsible for that utilization.

For instance, the CPU Profiler can correspond the utilization of a specific displayed resource at a given time with one or more functions of the application responsible for that utilization. The CPU Profiler allows the user to "inspect the details in recorded method traces, function traces, and system traces." https://developer.android.com/studio/profile/cpu-profiler (last visited 4/30/2024). Further, "[f]or each thread in your app process, you can find out which methods (Java) or functions (C/C++) are executed over a period of time and the CPU resources each method or function consumes during its execution." *Id.* These method and functions correspond to the application functions responsible for the CPU resource utilization.


You can use the CPU Profiler to inspect your app's CPU usage and thread activity in real time while interacting with your app, or you can inspect the details in recorded method traces, function traces, and system traces.

The detailed information that the CPU Profiler records and shows is determined by which recording configuration you choose:

- **System Trace:** Captures fine-grained details that allow you to inspect how your app interacts with system resources.
- **Method and function traces:** For each thread in your app process, you can find out which methods (Java) or functions (C/C++) are executed over a period of time, and the CPU resources each method or function consumes during its execution. You can also use method and function traces to identify *callers* and *callees*. A caller is a method or function that invokes another method or function, and a callee is one that is invoked by another method or function. You can use this information to determine which methods or functions are responsible for invoking particular resource-heavy tasks too often and optimize your app's code to avoid unnecessary work.

  When recording method traces, you can choose *sampled* or *instrumented* recording. When recording function traces, you can only use sampled recording.

https://developer.android.com/studio/profile/cpu-profiler (last visited 4/30/2024).

Below illustrates the CPU Profiler after recording a method trace, which illustrates the methods corresponding to specific CPU utilizations in the analysis pane (4).

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[H] correspond the utilization of a specific displayed resource at a given time with one or more functions of the application responsible for that utilization.

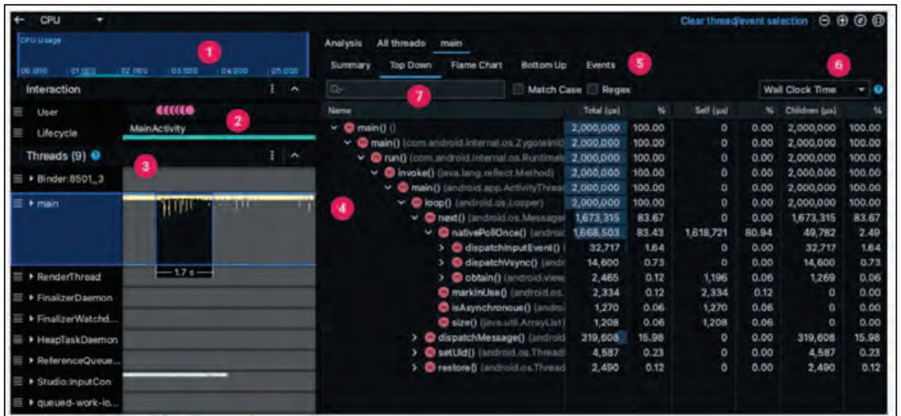


Figure 3. The CPU Profiler after recording a method trace.

https://developer.android.com/studio/profile/record-traces (last visited 1/17/2022).

The trace view of CPU Profiler allows users to view information from recorded traces. One such view is the Call Chart in the Threads timeline, as well as the Flame Chart, Top Down, Bottom Up, and Events tabs from the Analysis Pane. Below is an exemplary Call Chart view.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[H] correspond the utilization of a specific displayed resource at a given time with one or more functions of the application responsible for that utilization.

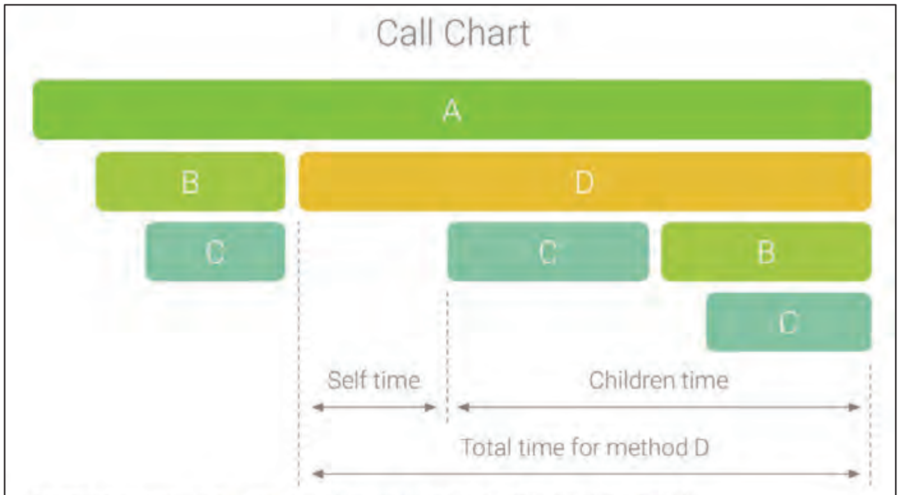
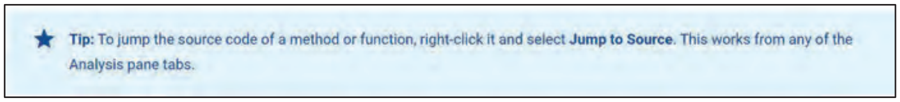


Figure 4. An example call chart that illustrates self, children, and total time for method D.

https://developer.android.com/studio/profile/inspect-traces (last visited 4/30/2024). And Android Studio provides the ability to jump to the source code for each of the methods displayed in the call chart for a recorded trace.


Tip: To jump the source code of a method or function, right-click it and select **Jump to Source**. This works from any of the Analysis pane tabs.

https://developer.android.com/studio/profile/inspect-traces (last visited 4/30/2024).

The Memory Profiler also allows the user to view where objects were allocated and when they were deallocated. Users can view "[t]he stack trace of each allocation, including in which thread" and "[w]hen the objects were deallocated."

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[H] correspond the utilization of a specific displayed resource at a given time with one or more functions of the application responsible for that utilization.

View memory allocations

Memory allocations show you *how* each Java object and JNI reference in your memory was allocated. Specifically, the Memory Profiler can show you the following about object allocations:

- What types of objects were allocated and how much space they use.
- The stack trace of each allocation, including in which thread.
- When the objects were *deallocated* (only when using a device with Android 8.0 or higher).

https://developer.android.com/studio/profile/memory-profiler (last visited 4/30/2024). The Call Stack tab shows where instances of memory objects were allocated and in what thread. Users can click "Jump to Source" to go to that code in the Android Studio editor.

To inspect the allocation record, follow these steps:

1. Browse the list to find objects that have unusually large heap counts and that might be leaked. To help find known classes, click the **Class Name** column header to sort alphabetically. Then click a class name. The **Instance View** pane appears on the right, showing each instance of that class, as shown in figure 3.
   - Alternatively, you can locate objects quickly by clicking **Filter**, or by pressing Control+F (Command+F on Mac), and entering a class or package name in the search field. You can also search by method name if you select **Arrange by callstack** from the dropdown menu. If you want to use regular expressions, check the box next to **Regex**. Check the box next to **Match case** if your search query is case-sensitive.
2. In the **Instance View** pane, click an instance. The **Call Stack** tab appears below, showing where that instance was allocated and in which thread.
3. In the **Call Stack** tab, right-click any line and choose **Jump to Source** to open that code in the editor.

https://developer.android.com/studio/profile/memory-profiler (last visited 4/30/2024).

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[H] correspond the utilization of a specific displayed resource at a given time with one or more functions of the application responsible for that utilization.

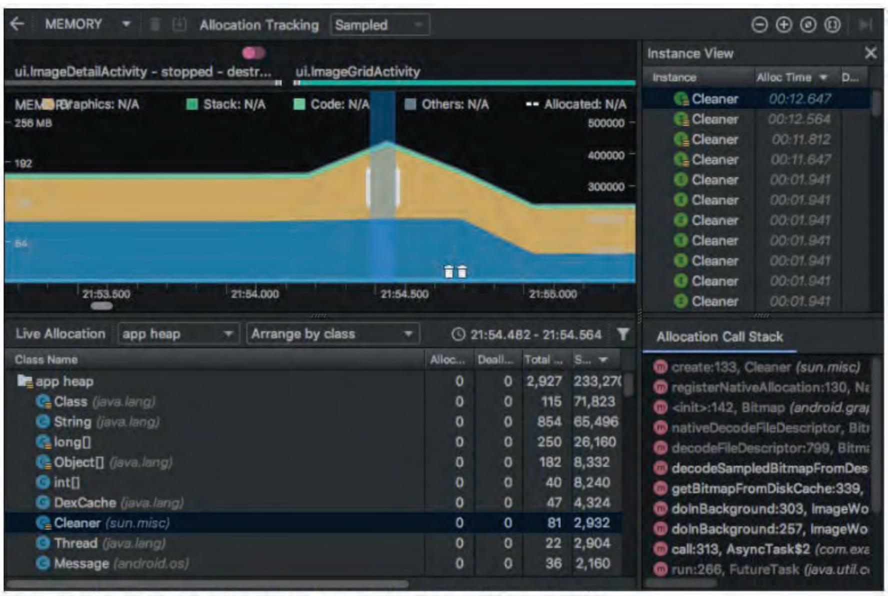


Figure 3. Details about each allocated object appear in the **Instance View** on the right

https://developer.android.com/studio/profile/memory-profiler (last visited 4/30/2024).

Additionally, the allocation and deallocation of JNI references can be viewed from the Memory Profiler. Using the JNI heap information, "you can find when and where global JNI references are created and deleted." https://developer.android.com/studio/profile/memory-profiler (last visited 4/30/2024).

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[H] correspond the utilization of a specific displayed resource at a given time with one or more functions of the application responsible for that utilization.

View global JNI references

Java Native Interface (JNI) is a framework that allows Java code and native code to call one another.

JNI references are managed manually by the native code, so it is possible for Java objects used by native code to be kept alive for too long. Some objects on the Java heap may become unreachable if a JNI reference is discarded without first being explicitly deleted. Also, it is possible to exhaust the global JNI reference limit.

To troubleshoot such issues, use the **JNI heap** view in the Memory Profiler to browse all global JNI references and filter them by Java types and native call stacks. With this information, you can find when and where global JNI references are created and deleted.

While your app is running, select a portion of the timeline that you want to inspect and select **JNI heap** from the drop-down menu above the class list. You can then inspect objects in the heap as you normally would and double-click objects in the **Allocation Call Stack** tab to see where the JNI references are allocated and released in your code, as shown in figure 4.

https://developer.android.com/studio/profile/memory-profiler (last visited 4/30/2024).

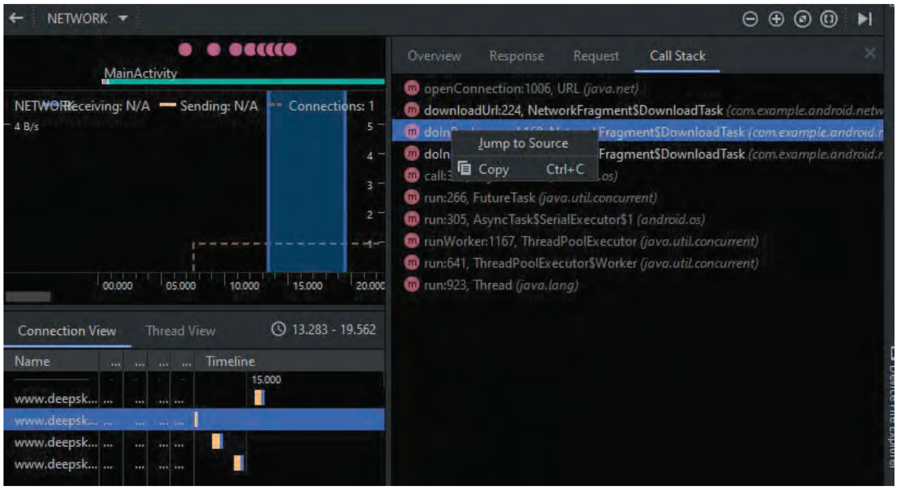
The Native Memory Profiler tracks allocations/deallocations of objects in native code for a specific time period. It can be arranged by callstack, permitting the user to find where in the code memory allocations/deallocations occur. *See* https://developer.android.com/studio/profile/memory-profiler (last visited 4/30/2024) (illustrating native allocations arranged by callstack).

The heap dump can also be viewed within Android studio to see "[w]here references to each object are being held in your code." https://developer.android.com/studio/profile/memory-profiler (last visited 4/30/2024).

The Network Profiler (and Network Inspector in newer Android versions) allows the user to view the call stack for network connections and threads, thus corresponding the network activity illustrated by the Network Profiler with functions of the application responsible for that activity. And, the user can "Jump to Source" to see the specific function listed in the call stack.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

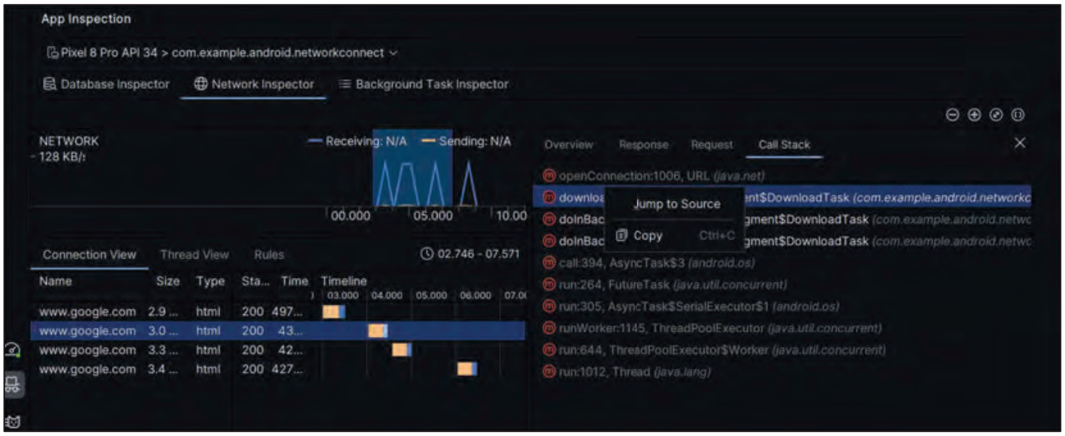
1[H] correspond the utilization of a specific displayed resource at a given time with one or more functions of the application responsible for that utilization.



Screenshot from Android Studio Arctic Fox: Network Profiler with Call Stack View.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

1[H] correspond the utilization of a specific displayed resource at a given time with one or more functions of the application responsible for that utilization.



Screenshot from Android Studio Iguana: Network Inspector with Call Stack View.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

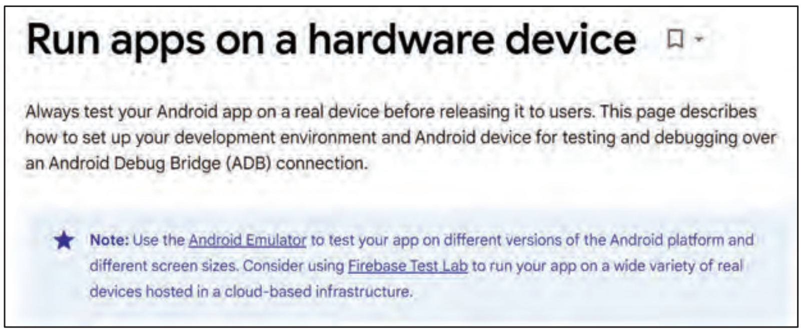
2[A] The medium of claim 1, wherein the instructions initiate transmission of the application that is being developed to one or more physical versions of a mobile device corresponding to the selected mobile device model.

Claim 2

2[A] The medium of claim 1, wherein the instructions initiate transmission of the application that is being developed to one or more physical versions of a mobile device corresponding to the selected mobile device model.

Android Studio includes instructions that initiate transmission of the application that is being developed to one or more physical versions of a mobile device corresponding to the selected mobile device model.

Android Studio's User's Guide explains the importance of testing applications on physical devices before publishing to users. And Android Studio is built to support transmission of applications under development to physical devices that correspond to the AVD model types discussed above (*see* 1[B]).



https://developer.android.com/studio/run/device (last visited 4/30/2024). To enable deployment to a physical device, the user must "open the Settings app [on the Android device], select Developer options, and then enable USB debugging (if applicable)." *Id.* Once this is done, you select Run in Android Studio to build and run your application on the physical device. This involves transmitting the application to the physical device for execution.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

2[A] The medium of claim 1, wherein the instructions initiate transmission of the application that is being developed to one or more physical versions of a mobile device corresponding to the selected mobile device model.

> **Connect to your device using USB**
>
> When you're set up and plugged in over USB, click **Run** in Android Studio to build and run your app on the device.

https://developer.android.com/studio/run/device (last visited 4/30/2024). Users can also transmit their applications to physical devices for execution using Wi-Fi in some instances.

> **Connect to your device using Wi-Fi**
>
> Android 11 and higher supports deploying and debugging your app wirelessly from your workstation via Android Debug Bridge (ADB). For example, you can deploy your debuggable app to multiple remote devices without physically connecting your device via USB and contending with common USB connection issues, such as driver installation.

https://developer.android.com/studio/run/device (last visited 1/17/2022).

4[A] The medium of claim 2, wherein the one or more characteristics indicative of the mobile device corresponding to the selected mobile device model include at least one of processor type, processor speed, storage access speed, RAM size, storage size, display width, display height, pixel depth, processor availability, RAM availability or storage availability.

Claim 4

4[A] The medium of claim 2, wherein the one or more characteristics indicative of the mobile device corresponding to the selected mobile device model include at least one of processor type, processor speed, storage access speed, RAM size, storage size, display width, display height, pixel depth, processor availability, RAM availability or storage availability.

See 1[C] (listing hardware profile properties and AVD Properties corresponding to a hardware profile for an Android Virtual Device).

For instance, the "Memory: RAM" hardware profile property and "Memory and Storage: RAM" AVD Property corresponds to RAM size and RAM availability. The "Memory and Storage" AVD Properties (*e.g.*, RAM, VM Heap, Internal Storage, SD Card) correspond to storage size and storage availability. The "Screen Size" and "Screen Resolution" hardware profile properties correspond to display width and display height. The "Emulated Performance: Multi-core CPU (Advanced)" AVD Property corresponds to the processor type and processor availability.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

5[A] The medium of claim 4, wherein the monitored resources include processor usage, RAM usage and network usage.

Claim 5

5[A] The medium of claim 4, wherein the monitored resources include processor usage, RAM usage and network usage.

See 1[F] (detailing the CPU Profiler for monitoring processor usage; the Memory Profiler for monitoring RAM usage; and the Network Profiler or Network Inspector for monitoring network usage).

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

8[A] The medium of claim 5, wherein the instructions simulate one or more network events that occur when interacting with a wireless network.

Claim 8

8[A] The medium of claim 5, wherein the instructions simulate one or more network events that occur when interacting with a wireless network.

Android Studio simulates one or more network events that occur when interacting with a wireless network.

For example, Android Studio can emulate/simulate the sending or receiving of a voice calls or SMS messages.

Send a voice call or SMS to another emulator instance

The emulator automatically forwards simulated voice calls and SMS messages from one instance to another. To send a voice call or SMS, use the dialer app or SMS app, respectively, from one of the emulators.

https://developer.android.com/studio/run/emulator-networking (last visited 4/20/2024).

To send an SMS message to another emulator instance:

1. Launch the SMS app, if available.
2. Specify the console port number of the target emulator instance as as the SMS address.
3. Enter the message text.
4. Send the message. The message is delivered to the target emulator instance.

https://developer.android.com/studio/run/emulator-networking (last visited 4/20/2024).

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

8[A] The medium of claim 5, wherein the instructions simulate one or more network events that occur when interacting with a wireless network.

To initiate a simulated voice call to another emulator instance:

1. Launch the dialer app on the originating emulator instance.
2. As the number to dial, enter the console port number of the target instance.

   You can determine the console port number of the target instance by checking its window title, if it is running in a separate window, but not if it is running in a tool window. The console port number is reported as "Android Emulator (<port>)".

   Alternatively, the `adb devices` command prints a list of running virtual devices and their console port numbers. For more information, see Query for devices.
3. Click the dial button. A new inbound call appears in the target emulator instance.

https://developer.android.com/studio/run/emulator-networking (last visited 4/20/2024).

Telephony emulation is supported by the "gsm" and "cdma" console commands.

Telephony emulation	Description
The Android emulator includes its own GSM and CDMA emulated modems that let you simulate telephony functions in the example, with GSM you can simulate inbound phone calls and establish and terminate data connections. With CDMA you p subscription source and the preferred roaming list. The Android system handles simulated calls exactly as it would actual (emulator does not support call audio.	

https://developer.android.com/studio/run/emulator-console#telephony (last visited 4/30/2024). As illustrated below, the "gsm" command can emulate/simulate an inbound phone call, accept and inbound phone call, terminate inbound/outbound calls, and produce a busy signal.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

8[A] The medium of claim 5, wherein the instructions simulate one or more network events that occur when interacting with a wireless network.

gsm {call|accept|cancel|busy} *phonenumber*	The gsm parameters are the following: • call: Simulate an inbound phone call from *phonenumber*. • accept: Accept an inbound call from *phonenumber* and change the call state to active. You call state to active only when its current state is waiting or held. • cancel: Terminate an inbound or outbound phone call to or from *phonenumber*. • busy: Close an outbound call to *phonenumber* and change the call state to busy. You can cha state to busy only when its current state is waiting.

https://developer.android.com/studio/run/emulator-console#telephony (last visited 4/30/2024).

Emulation of SMS-based network events are provided by the "sms" console command.

SMS emulation	Description
sms send *sender-phone-number* *textmessage*	Generate an emulated incoming SMS. The following list describes the parameter and their values: • *sender-phone-number*: Contains an arbitrary numeric string. • *textmessage*: The sms message. The following example sends the message "hi there" to the 4085555555 phone number. `sms send 4085555555 hi there` The console forwards the SMS message to the Android framework, which passes it to an app on tl that handles SMS, such as the Messages app. If you pass 10 numbers, the app formats it as a pho Longer or shorter numeric strings display the way you sent them.

https://developer.android.com/studio/run/emulator-console#sms (last visited 4/30/2024).

<u>Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)</u>

9[A] A non-transitory, computer-readable medium comprising software instructions for developing an application to be run on a mobile device, wherein the software instructions, when executed, cause a computer to:

Claim 9

9[A] A non-transitory, computer-readable medium comprising software instructions for developing an application to be run on a mobile device, wherein the software instructions, when executed, cause a computer to:

See 1[A].

<u>**Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)**</u>

9[B] display a list of a plurality of mobile devices from which a user can target a particular device;

9[B] display a list of a plurality of mobile devices from which a user can target a particular device;

See 1[B] (detailing a list of mobile device models from which a user can select). Because the mobile device models can be selected by the user, the user can target a particular mobile device by selecting its corresponding mobile device model.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

9[C] model one or more characteristics indicative of the targeted mobile device;

9[C] model one or more characteristics indicative of the targeted mobile device;

See 1[C] (detailing how each mobile device model includes one or more characteristics indicative of a corresponding mobile device). Because each mobile device model includes characteristics indicative of its corresponding mobile device, the mobile device model models these characteristics. In particular, these characteristics are modeled when the mobile device model (or AVD) is run by Android Emulator. *See* 1[D]-1[E] (detailing how AVDs and their properties, or characteristics, are simulated/emulated by Android Emulator).

9[D] monitor utilization of a plurality of resources over time as the application is running;

See 1[F].

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

9[E] display simultaneously two or more graphical images of the application's resource utilization as it is running, wherein each graphical image relates to a different resource and is synched in time as the application is running;

9[E] display simultaneously two or more graphical images of the application's resource utilization as it is running, wherein each graphical image relates to a different resource and is synched in time as the application is running;

See 1[G].

The Profiler view in Arctic Fox and earlier versions of Android Studio shows at least five graphical images (CPU, Memory, Network Sending, Network Receiving, Energy) of the application's resource utilization as it is running. *See* https://developer.android.com/studio/profile/android-profiler (last visited 1/17/2022) ("The Android Profiler tools provide real-time data to help you to understand how your app uses CPU, memory, network, and battery resources."); *id.* ("Use the zoom buttons to control how much of the timeline to view, or ***use the Attach to live button to jump to the real-time updates***." (emphasis added)).

The Profiler view in Bumblebee and later versions of Android Studio shows at least 2 graphical images (CPU and Memory) of the applications' resource utilization as it is running.

As the screenshots below show, the Profiler views are synched in time as the application is running. The timeline runs across the bottom of the Profiler screen, and the line across shows the moment in time for each of the individual profiler displays (CPU, Memory, Network Sending, Network Receiving, Energy).

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

9[E] display simultaneously two or more graphical images of the application's resource utilization as it is running, wherein each graphical image relates to a different resource and is synched in time as the application is running;

Screenshot from Android Studio Arctic Fox: Profiler View (showing CPU% at 13:35.429).

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

9[E] display simultaneously two or more graphical images of the application's resource utilization as it is running, wherein each graphical image relates to a different resource and is synched in time as the application is running;

Screenshot from Android Studio Arctic Fox: Profiler View (showing Memory at 14:15.023).

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

9[E] display simultaneously two or more graphical images of the application's resource utilization as it is running, wherein each graphical image relates to a different resource and is synched in time as the application is running;

Screenshot from Android Studio Arctic Fox: Profiler View (showing Sent Rate and Received Rate at 14:38.411).

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

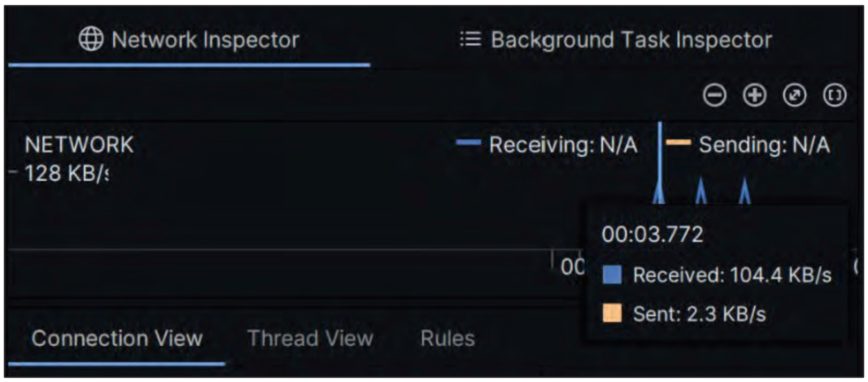
9[E] display simultaneously two or more graphical images of the application's resource utilization as it is running, wherein each graphical image relates to a different resource and is synched in time as the application is running;

Screenshot from Android Studio Arctic Fox: Profiler View (showing Energy at 14:57.331).

As the screenshot below shows, the Network Inspector view is synched in time as the application is running, providing synchronized display of both "Received" data and "Sent" data.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

9[E] display simultaneously two or more graphical images of the application's resource utilization as it is running, wherein each graphical image relates to a different resource and is synched in time as the application is running;



Screenshot from Android Studio Iguana: Network Inspector View (showing Received and Sent at 3.772).

<u>**Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)**</u>

9[F] identify one or more functions of the application responsible for utilization of a specific displayed resource at a given time.

9[F] identify one or more functions of the application responsible for utilization of a specific displayed resource at a given time.

See 1[H] (detailing how the profiler displays and network inspector displays correlate specific points in time with one or more functions of the application responsible for the utilization depicted in the display, *e.g.*, CPU, Memory, Network Send Rate, Network Receive Rate).

<u>Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)</u>

22[A] A non-transitory, computer-readable medium comprising software instructions for developing an application to be run on a mobile device, wherein the software instructions, when executed, cause a computer to:

Claim 22

22[A] A non-transitory, computer-readable medium comprising software instructions for developing an application to be run on a mobile device, wherein the software instructions, when executed, cause a computer to:

See 1[A].

22[B] simulate one or more characteristics indicative of the mobile device, wherein the one or more characteristics indicative of the mobile device include at least one of processor type, processor speed, storage access speed, RAM size, storage size, display width, display height, pixel depth, processor availability, RAM availability or storage availability;

See 1[D] (detailing simulation of mobile device characteristics); 4[A] (explaining how those characteristics relate to processor type, RAM size, storage size, display width, display height, processor availability, RAM availability, and storage availability).

22[C] monitor utilization of a plurality of resources over time as the application is running, wherein the monitored resources include at least one of processor usage and RAM usage;

See 1[F] (detailing CPU Profiler and Memory Profiler).

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

22[D] display one or more graphical images of the application's resource utilization;

22[D] display one or more graphical images of the application's resource utilization;

See 1[G].

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

22[E] correspond the utilization of a specific displayed resource at a given time with one or more functions of the application responsible for that utilization;

22[E] correspond the utilization of a specific displayed resource at a given time with one or more functions of the application responsible for that utilization;

See 1[H].

22[F] initiate transmission of the application that is being developed to one or more physical versions of the mobile device.

22[F] initiate transmission of the application that is being developed to one or more physical versions of the mobile device.

See 2[A].

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

24[A] The medium of claim 22, wherein the instructions simulate one or more characteristics, including bandwidth, indicative of a network on which the mobile device can operate.

Claim 24

24[A] The medium of claim 22, wherein the instructions simulate one or more characteristics, including bandwidth, indicative of a network on which the mobile device can operate.

See 1[E] (detailing the simulation of network bandwidth).

[26] The medium of claim 24, wherein the instructions simulate one or more network events that occur when interacting with a wireless network, wherein a user can create scripts to emulate actions of real user behavior to determine the performance of the application, or the network, or both.

Claim 26

[26] The medium of claim 24, wherein the instructions simulate one or more network events that occur when interacting with a wireless network, wherein a user can create scripts to emulate actions of real user behavior to determine the performance of the application, or the network, or both.

See 8[A] (detailing simulation of network events).

Android Studio is configured to allow users to create scripts that can emulate actions of real user behavior to determine the performance of the application, or the network, or both. Android Studio supports the creation of local unit tests and instrumented tests.

> **Test types and locations**
>
> The location of your tests depends on the type of test you write. Android projects have default source code directories for local unit tests and instrumented tests.

https://developer.android.com/studio/test/test-in-android-studio (last visited 5/1/2024).

Local unit tests test components of the application source code and therefore impact the performance of the application.

> **Local unit tests** are located at `module-name/src/test/java/`. These are tests that run on your machine's local Java Virtual Machine (JVM). Use these tests to minimize execution time when your tests have no Android framework dependencies or when you can create test doubles for the Android framework dependencies. For more information on how to write local unit tests, see Build local unit tests.

https://developer.android.com/studio/test/test-in-android-studio (last visited 5/1/2024).

Instrumented tests run on the target device or an emulator of a target device. These tests "let you control the app under tests from your test code" and can be used to "automate user interaction."

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

[26] The medium of claim 24, wherein the instructions simulate one or more network events that occur when interacting with a wireless network, wherein a user can create scripts to emulate actions of real user behavior to determine the performance of the application, or the network, or both.

Instrumented tests are located at `$module-name/src/androidTest/java/`. These tests run on a hardware device or emulator. They have access to `Instrumentation` APIs that give you access to information, such as the `Context` class, on the app you are testing, and let you control the app under test from your test code. Instrumented tests are built into a separate APK, so they have their own `AndroidManifest.xml` file. This file is generated automatically, but you can create your own version at `$module-name/src/androidTest/AndroidManifest.xml`, which will be merged with the generated manifest. Use instrumented tests when writing integration and functional UI tests to automate user interaction, or when your tests have Android dependencies that you can't create test doubles for. For more information on how to write instrumented tests, see Build instrumented tests and Automate UI tests.

https://developer.android.com/studio/test/test-in-android-studio (last visited 5/1/2024).

Android Studio creates the project structure and sample tests to support both local unit tests and instrumented tests.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

[26] The medium of claim 24, wherein the instructions simulate one or more network events that occur when interacting with a wireless network, wherein a user can create scripts to emulate actions of real user behavior to determine the performance of the application, or the network, or both.

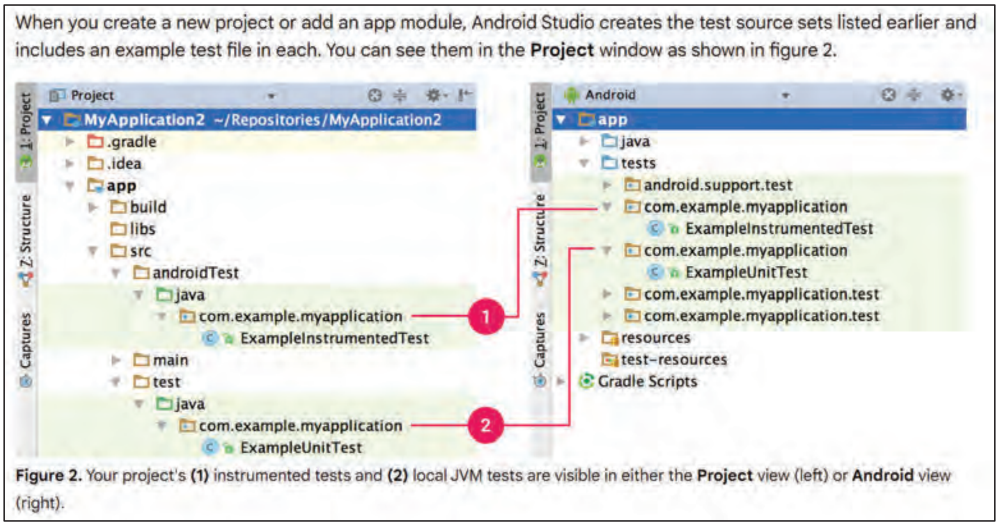

When you create a new project or add an app module, Android Studio creates the test source sets listed earlier and includes an example test file in each. You can see them in the **Project** window as shown in figure 2.

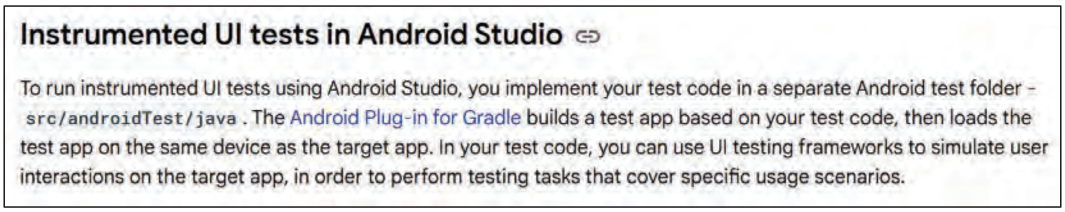
Figure 2. Your project's **(1)** instrumented tests and **(2)** local JVM tests are visible in either the **Project** view (left) or **Android** view (right).

https://developer.android.com/studio/test/test-in-android-studio (last visited 5/1/2024). These tests impact the performance of the application under tests and/or network and can emulate the actions of real user behavior.

Android Studio also supports UI tests which are scripts that emulate actions of real user behavior to determine the performance of the application, or the network, or both.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

[26] The medium of claim 24, wherein the instructions simulate one or more network events that occur when interacting with a wireless network, wherein a user can create scripts to emulate actions of real user behavior to determine the performance of the application, or the network, or both.



Instrumented UI tests in Android Studio

To run instrumented UI tests using Android Studio, you implement your test code in a separate Android test folder - `src/androidTest/java`. The Android Plug-in for Gradle builds a test app based on your test code, then loads the test app on the same device as the target app. In your test code, you can use UI testing frameworks to simulate user interactions on the target app, in order to perform testing tasks that cover specific usage scenarios.

https://developer.android.com/training/testing/instrumented-tests/ui-tests (last visited 5/2/2024). Android supports several APIs for writing UI test scenarios and scripts.

Infringement Chart for U.S. Patent No. 10,353,811 (Google's Android Studio Tools)

[26] The medium of claim 24, wherein the instructions simulate one or more network events that occur when interacting with a wireless network, wherein a user can create scripts to emulate actions of real user behavior to determine the performance of the application, or the network, or both.

Jetpack frameworks

Jetpack includes various frameworks that provide APIs for writing UI tests:

- The **Espresso testing framework** (Android 4.0.1, API level 14 or higher) provides APIs for writing UI tests to simulate user interactions with *Views* within a single target app. A key benefit of using Espresso is that it provides automatic synchronization of test actions with the UI of the app you are testing. Espresso detects when the main thread is idle, so it is able to run your test commands at the appropriate time, improving the reliability of your tests.
- **Jetpack Compose** (Android 5.0, API level 21 or higher) provides a set of *testing APIs* to launch and interact with Compose screens and components. Interactions with Compose elements are synchronized with tests and have complete control over time, animations and recompositions.
- **UI Automator** (Android 4.3, API level 18 or higher) is a UI testing framework suitable for cross-app functional UI testing across system and installed apps. The UI Automator APIs allows you to perform operations such as opening the Settings menu or the app launcher on a test device.
- **Robolectric** (Android 4.1, API level 16 or higher) lets you create *local* tests that run on your workstation or continuous integration environment in a regular JVM, instead of on an emulator or device. It can use Espresso or Compose testing APIs to interact with UI components.

https://developer.android.com/training/testing/instrumented-tests/ui-tests (last visited 5/2/2024).